LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN
510.84
Il6r
no.782-787
cop. 2

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

JUN 6 1973

MAY 9 RECD

L161—O-1096

Digitized by the Internet Archive
in 2013

http://archive.org/details/measuringknowled786mich

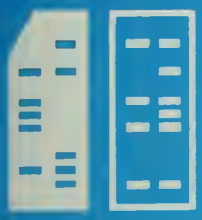
IL6N
UIUCDCS-R-76-786
no.786

MEASURING THE KNOWLEDGE-CONTENT OF PROGRAMS

by

Donald Michie

May 1976



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

The Library of the
JUL 14 1976
University of Illinois
at Urbana-Champaign

UIUCDCS-R-76-786

MEASURING THE KNOWLEDGE-CONTENT OF PROGRAMS

by

Donald Michie

May 1976

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CONTENTS PAGE

Introductory outline 1

Basic definition: f 3

Basic definition: K 4

Basic definition: L 4

Basic definition: e 4

Purity of formulations 4

Properties of p in relation to f 5

Motivation and informal remarks 6

Technical need 7

Assumptions and restrictions 8

Information about a function 9

Knowledge of a function 10

Partitions of memory 11

The predicate "known" 12

Knowledge in a heuristic 13

ω-concepts and β-concepts 14

Simplified and refined forms 15

Simplified theory: definitions 15

Information concerning f 15

Knowledge-content of the look-up program p_T 16

Knowledge-content of the solution program p_A 16

Difference between information-content and knowledge-content . 17

Knowledge-content of advice program and knowledge-overhead of control program 18

Refinements: partial (truncated) evaluation 21

CONTENTS (continued) PAGE

Refinements: error 23

Simplest case of error: no gradations, no partial evaluation 24

Gradations but no partial evaluation 24

Partial evaluation, no gradations 25

General case: gradations of error in partial evaluation . . . 25

Refinements: differential argument-utilities 28

Some β-independent quantities 29

The memo-function as a model of rote-learning 30

Telegram-factory model of concept learning 33

Learning, rote-skill and grasp 36

Concluding remarks 39

Acknowledgments 40

References 41

Appendix 1: Relation to Aristotle's theory of knowledge 49

Reference 51

Appendix 2: What to substitute for S 52

Reference 52

Introductory outline

An approach is described for defining and numerically measuring a system's knowledge of a finite function f and for estimating the net knowledge-increment obtainable from a given body of advice. The key idea is that a system "knows" only what it can evaluate (whether by calculation or by look-up) in user-acceptable time. The role of advice is to make it know more. Knowledge of f is expressed quantitatively in information-bits (not to be confused with store-bits). If β denotes an upper bound to the time available for evaluating f for any argument, the information-content of what is β-evaluable is the amount of knowledge, K, about f.

The envisaged application is to the evaluation of a recursively defined function by an algorithm A implemented as a program p_A to which a body of advice B is made available in the form of a knowledge-base p_B, through the mediation of a control program p_C. p_B is a kind of data-base (parts of which may on occasion be executed as program). A typical action of the advice contained in it is to alter p_A's termination condition in such a way as to cause earlier exit from the recursion without altering the computed result.

Advice theory offers a cost-benefit analysis of p_B and of its constituents. Benefit is expressed in units of knowledge. Cost is attributed to store occupancy and is expressed in store-bits. All measures are defined with respect to the machine specification S and the solution program p_A. The measured benefit-to-cost ratio for p_A alone is termed the system's algorithmic advantage. The measured net knowledge-increment conferred by p_B divided by its store-occupancy is the system's advisory advantage relative to p_A. The system's computational advantage is the

total knowledge (algorithmic plus advisory less an overhead associated with the operations of p_C) divided by the store-cost (store occupancy of p_A plus store occupancy of p_B plus store occupancy of advice-management routines p_C). The term penetration is used when advantage is expressed on a logarithmic scale. Computational yield is defined as the ratio of the actual amount of knowledge to the maximum attainable with the given S and store-cost; the usefulness of this measure is restricted to the case where the store available is less than or equal to that which just permits complete knowledge of f (i.e. less than or equal to f's β-complexity, see below). More versatile measures are effectiveness, defined as the ratio of the actual to the maximum advantage, and efficiency, defined as the ratio of the minimum to the actual store occupancy for given K.

Knowledge-increase is seen as occurring not only through input of advice from the user but also through the system's storage of instances of, or (more powerfully) inductive generalization over, past evaluations of f. These processes can all modify or add to p_B so as to increase its knowledge-content. They can also, through restructuring or deleting existing advice, reduce p_B's store occupancy. In either case advisory advantage is increased through purely internal reorganization. We thus have a domain-independent way of measuring knowledge-acquisition in machine learning, following a scheme of which the memo-function (Michie, 1967) is a primitive example.

By a restriction of Solomonoff-Kolmogorov complexity we obtain β-complexity as a numerical measure of a problem-domain's difficulty. Extensions of advice theory deal with partial evaluation, evaluation

error, and differential utilities of evaluating f for different arguments. With this last extension in place of "pure" knowledge we speak of "useful" knowledge. Computational advantage and penetration as earlier defined divide correspondingly into "pure" and "useful" forms (each consisting as before of algorithmic and advisory parts), and similarly for yield, efficiency and any other quantities compounded from the primitive definitions.

The rate at which p (p_A, or p_A plus p_B plus p_C, as the case may be) can deliver complete information about f is independent of β, and is termed its computational capacity, expressed in information-bits/sec. Dividing by the bit-cost of p gives its computational power - the number of store-equivalents of information about f delivered per second. This is a measure of a system's overall performance in its knowledge-delivery as opposed to knowledge-acquisition aspect. The latter is treated by an extension of the theory addressed to measures of the performance of rote-learning and inductive generalization programs.

Two instances of f, one numerical, and one non-numerical, are mentioned, namely finite-factorial, and bestmove (in chess). Reference is also made to a probabilistic generalization by I. J. Good (1976) of the main definition.

In anticipation of the body of the paper we give below a brief snapshot of the primitive notions.

Basic definition: f

f: $X \rightarrow Y$, where X is an ordered domain of size N and Y, the co-domain, is of size M.

Basic definition: K

A program p's "useful knowledge" about a finite function f is measured in information-bits as

$$K = \sum_{i=1}^{N} u_i \times \text{surprisal}\ (Y_i)$$

where u_i is the normalized frequency with which p is invoked to evaluate f for the i-th member of X, and Y_i is the candidate set remaining after evaluating $f(x_i)$, using p, for just β seconds. Surprisal (Samson 1951) is $\log_2 \frac{1}{p_i}$ where p_i is the prior probability that an arbitrarily chosen member of Y will fall into Y_i. Evaluation is assumed to have the form of a successive approximation so that $Y \rightarrow Y_{i,1} \rightarrow Y_{i,2} \ldots$ in successive cycles, 0, 1, 2, ... of the computation.

Basic definition: L

L is the number of store-bits occupied by the program p. L_K is the bit-cost of a program containing K bits of useful knowledge. L_K^* is the minimum possible L for any such program: L_β is the minimum possible L for a program containing complete knowledge, i.e. K = I(f). K_L^* is the maximum possible K for any program of no more than L store-bits. I(f) is the total information-content of f, calculated in a way to be described later.

Basic definition: e

The extraction time, e, is the time needed by the system to evaluate f fully over the whole of its domain.

Purity of formulations

We do not propose to make a fetish of purity, in particular when it comes to quarantining abstract objects such as "algorithms", "arguments"

and the like from their counterparts in the world of concrete embodiments - such things as "programs" and "inputs". We have taken the trouble above to write " ... frequency with which p is invoked to evaluate f for the i-th member of X, ... " when we could easily have written " ... frequency with which p is applied to the i-th member of X, ... " (strictly speaking the latter would be wrong: one cannot apply a concrete object to an abstract one). But we do not undertake to observe these laws of quarantine on occasions when law-breaking offers large gains of brevity or perspicuousness. If mathematical taste is thereby offended we are sorry. But communication with the reader who is willing to "see what we mean" has, rightly or wrongly, been placed first.

Properties of p in relation to f:

	Units of measurement	
Information-content (of f)	Information-bits	$I(f)$
Knowledge (possessed by p about f)	Information-bits	K
Store-cost	Store-bits	L
Computational advantage	Pure number	$C = \frac{K}{L} = \frac{K_A + K_B + K_C}{L_A + L_B + L_C}$
Penetration	Decs	$D = \log_{10} K - \log_{10} L$
Yield	Pure number	$\frac{K}{K^*_L}$
Efficiency	Pure number	$\frac{L^*_K}{L_K}$
Effectiveness	Pure number	$\frac{K \times L_\beta}{L \times I(f)}$

Advisory advantage	Pure number	$C_B = \frac{K_B}{L_B} = \frac{K_r + K_c}{L_r + L_c}$
Advisory penetration	Decs	$D_B = \log_{10} K_B - \log_{10} L_B$
Operating cost	Bit-seconds	Lt
Learning index	Decs/bit-seconds	$\frac{\Delta D_B}{L_U t_U}$

K and L denote the knowledge-content and bit-cost of a given p and L_β is the β-complexity of f, i.e. the minimum bit-cost for a program possessing total knowledge of f, I(f) in amount. L^*_K is the β-complexity of that subset of f covered by K, i.e. of the partial function f_K. In the index of learning L_U denotes the store-cost and t_U the processor time associated, when running the program (p_A, p_B, p_C, p_U), with p_U, a procedure which uses results computed by (p_A, p_B, p_C) to update p_B. Subscripts r and c denote "rote" and "concept" respectively, distinguishing two forms in which information about f may be stored in p_B.

It is not expected that the foregoing "snapshot" will be immediately intelligible: the rest of the paper can be regarded as an attempt to make it so.

Motivation and informal remarks

In AI we talk about "knowledge", but would like to progress beyond the mere slogan.

Is "knowledge" that which is explicitly stored? At first sight this seems reasonable. For example, it takes care of the following kind of paradox. A number theorist knows a great deal more than he did when he

first learned the axioms of arithmetic. Yet everything he knows, and infinitely more besides, is strictly deducible from these axioms. Therefore, there cannot be anything new for him to have gained. What, then, is this extra stuff which he knows? Presumably it consists of explicitly stored conclusions which previously were only implicitly available to him. Since everything that number theory has done with the integers since God made them has been strictly redundant, we conclude that the only new value created has been that of accessibility. The point of new knowledge is to make less time-consuming the retrieval of old truth.

According to the explicit/implicit idea, I know the definition of a prime number and I know that 23 is prime. But I do not know that 233 is prime, although I can quickly work it out.

At second sight, the criterion of explicit storage is not quite so reasonable. We suggest that the notion of explicit storage is handy only because it correlates highly (in most information processing systems) with the speed with which the needed information can be retrieved.

On this more general criterion a desk-top calculator which calculates factorials in a second has as much claim to a "knowledge" of the function as a similarly priced table-look-up machine. This is essentially the Aristotelian position (see Appendix 1) and we shall adopt it here. Knowledge must be of something, and for definiteness it will always be of a finite mathematical function.

Technical need

Information theory allows us to specify the quantity of information contained in a message and to calculate the least space in a computer memory required to store a minimal encoding of it.

Computation theory allows us to say what functions are computable in principle, and whether particular programs compute them.

Complexity theory can show, for the evaluation of certain simple functions, how some measures of computational cost can be minimized.

What we lack is a calculus which will enable us to say:

not merely how much information about a function f is stored in a computing system;

not merely whether and how the system could in principle compute f;

not merely how the computation might be optimized with respect to specified cost criteria;

but how much knowledge of f the system possesses.

"Knowledge" should be defined in such a way as to be compatible with the informal use of the word in the machine intelligence literature. The definition, and any calculus based upon it, must be capable of characterizing the dynamical aspects of knowledge, its acquisition, transfer, and use. The essence of our approach will be to regard knowledge as the practically retrievable part of the total information possessed by the system about f.

Assumptions and restrictions

We measure information in the normal way, and express it in bits. In order to avoid confusion we shall use the word "store-bit" to mean a binary digit in machine memory.

We exclude from strict consideration any function (e.g. a real-valued function or a function of a real variable) which may require infinitely many bits to represent a value or an argument. But we do not object to finite functions by which such functions may be approximated.

Examples (the first chosen for its simplicity and the second for its complexity):

finite-factorial: $X \longrightarrow Y$

$x \leq xmax$ and $x \in$ non-neg. integers $\qquad y \in$ positive integers

bestmove: legal chess-positions → legal moves

Consider a member of this class, f: $X \rightarrow Y$. Our approach is to regard "information" as a measure of certain properties of a static representation of f, conceived simply as a message which encodes f. In this classical information-theoretic world there are no store-time trade-offs, and the information-content of a message is equal to the minimal number of bits required to store it explicitly. By contrast, "knowledge" measures certain properties of the evaluation of f, whether by look-up from store, by calculation, or by any mixture of the two. In this world we can, and typically do, encode long messages in short programs. How much do such programs "know"?

Information about a function

We picture f as laid out in extensional (tabular) form, comprising a sequence of (x, y) pairs, N in number. The first member of each pair is drawn from the ordered domain X, and the second from the co-domain Y. A message of length N, where N = size (X), has thus been constructed, the values $y_1, y_2, y_3, \ldots, y_N$ being regarded as a sequence of choices from an alphabet Y. Assuming equiprobable choice, the information content of such a message is given by Hartley's (1927) expression $N \log_2 M$, where M = size (Y). This follows from the fact that there are exactly M^N possible functions which could be constructed with the given domain and co-domain. Allowing for different frequencies of the different y-values,

we have $-\sum_{j=1}^{M} n_j \log_2 \frac{n_j}{N}$ where n_j is the number of occurrences of the j-th member of Y. An equivalent, and for our purposes more convenient, formulation is obtained by summing over the domain rather than the co-domain of f, thus: $-\sum_{i=1}^{N} \log_2 \frac{n_i}{N}$.

Suppose, now, that only k of the n entries have been inscribed in the table. Then the table's information content is $-\sum_{i=1}^{k} \log_2 \frac{n_i}{N}$, where n_i is the number of occurrences in the complete table of (x, y) pairs whose right-hand element matches that of the i-th entry of the reduced, k-sized table. This expression is the basis for measuring the information-content of a "partially known" function.

Knowledge of a function

The test of "knowledge" is the ability to answer questions: "What is the finite-factorial of 6?", "What is the best continuation of this chess position?" If an answer can be produced within some time limit, then the system is held to "know" it.

As preamble to an informal definition, consider the respective tasks of answering

(1) "What is the finite-factorial of 6?", and

(2) "What is the finite-factorial of 10^6?"

In the second case it would seem that since the answer is so much harder to get, there must be more information obtained. This is not so. An answer conveys $-\log_2 \frac{n_i}{N}$ bits of information (its "surprisal"), and that is the end of the matter. But it is true that either

(i) the system is more likely to deliver an answer to (1) than to (2) within an acceptable waiting time (our definition of knowledge will allow us to say that the system is more likely to "know" (1) than (2)), or

(ii) if obliged to reply within a time limit, the system will be able to answer (1) to a higher degree of approximation than (2). A definition which permitted degrees of knowledge of f(x) would allow us to say that the system has more knowledge of (1) than of (2). Such a definition is possible and has interesting consequences. These are discussed later.

Partitions of memory

We subdivide machine memory into six sectors, dedicated respectively to the following purposes:

1. Storage of the input argument ("data space").
2. Storage of a solution program p_A for the function, f, ("function space").
3. Storage of data generated and used by the evaluation algorithm in the course of, and solely for the purposes of, the given evaluation. This "work-space" is empty on entry and cleared on exit.
4. Storage of an advice program p_B, (the "knowledge base" or "body of advice"), which modifies the action of p_A, typically by causing earlier exit without affecting the computed result. We refer to this sector of memory as "advice space".
5. Storage of a control program p_C for making p_B available to p_A.

6. Storage of an update program p_U for reorganizing p_B between evaluations, at the lowest level by rote-operations, at the highest by inductive generalization.

In most conventional computing, machine memory is partitioned among 1, 2, and 3 only. Our view of the distinguishing characteristics of an intelligent system is that the key components are precisely the last three. Systems exhibiting such a morphology as the above are at the dawn of their evolution. Specimens are scarce, and highly evolved specimens non-existent. The best known example is probably Buchanan, Feigenbaum and Lederberg's Heuristic DENDRAL (for recent reviews see Michie and Buchanan, 1974, Buchanan, 1975). A system of the same kind for chess end-games would map onto the six sectors as follows:

1. Current board position.
2. Lookahead routine (every chess program has one).
3. Lookahead tree.
4. Body of advice, consisting of chess heuristics to guide and prune the lookahead routine.
5. Routines to access and apply the heuristics.
6. A learning routine using accumulating samples of the extensional representation of bestmove, as they are discovered by the lookahead, to add new heuristics and modify old ones.

The predicate "known"

A predicate known has as arguments a function f, an input argument x, a machine specification S (see Appendix 2), programs p_A, p_B and p_C corresponding to 2, 4 and 5 above, and a time bound β. known (f, x, S, p_A, p_B, p_C, β) is true iff f(x) can be evaluated in no more than

β time units. If, in the given advice state f(x) is known for all x, then we say that f is β-evaluable on the given machine by the given (p_A, p_B, p_C). If f(x) is known only for some x we speak of the "known part" of the function. There is thus an extension here of the familiar notion of a partially defined function, except that we deal with partially known functions. In the new system, which incorporates the time dimension, "known" corresponds to a subset of the concept "defined" in the old system, but unlike "defined" changes dynamically. Thus, a system's knowledge of a function is the information-content of its β-evaluable extent.

We accordingly say that the number of units of knowledge of f is equal to the minimal number of bits of information required to specify that part of f which is β-evaluable, i.e., how much is known.

Knowledge in a heuristic

Consider (by way of example) Huberman's (1968) treatment of chess programming in certain elementary end-games, where p_A stands for her "forcing tree" lookahead routine and p_B stands for her logic formulae expressing "better" and "worse" predicates. We wish to measure the knowledge-content of individual heuristics in p_B. Proceed as follows (in principle: in practice we would sample from the domain, not chart it exhaustively).

1. With the heuristic loaded, run p_A for each x in X, and record the number of bits of information corresponding to the known part of the function.
2. Unload the heuristic and repeat.
3. The difference between the two measurements gives the amount of knowledge about f present in the heuristic.

The knowledge-gain has to be traded against the heuristic's storage cost:

the ratio of gain (in information-bits) to cost (in store-bits) is the advisory advantage.

ω-concepts and β-concepts

We are turning our backs on "in principle" concepts of computability, completeness, contradiction and so forth. In so far as we need to refer to them we can speak of "ω-computability," "ω-completeness," "ω-contradiction," etc. Instead of these concepts we envisage a calculus of "β-concepts." A system is β-complete if every question admissible within the system can be answered in the given state of the advice in no more than β time units; a set of assertions is β-inconsistent if a contradiction can be β-derived and β-consistent if it cannot (a β-consistent body of assertions may be ω-consistent or it may be ω-inconsistent); an assertion is β-redundant if its addition leaves unaffected what can be β-derived. The theory evidently has implications for the special case of function evaluation known as automatic deduction, where the concept of a β-theorem presents itself and claims relationship to what is usually called "belief" (see Meltzer 1970 for the notion of logical validity subject to a depth limit on the deduction).

We do not worry at this stage about the exact magnitude of the bound β. For concreteness we can suppose that the system must answer a question, if it can at all, within a few seconds or minutes rather than milli-seconds or days and that it has access to a store of say a million, rather than a thousand or a billion, locations, with access times in the microsecond rather than the millisecond range. Design and implementation of an "advice language" would be desirable, with facilities for handling and coordinating all of Sectors 1 - 6 above. But the case for developing and testing the theoretical ideas is independent of these possibilities.

Simplified and refined forms

An account of the theory follows. First a simplified form is presented from which all issues likely to distract from the central exposition are omitted - specifically the treatment of truncations, errors and utilities. These elaborations are then introduced to complete the theory. A bare framework of some elementary objects and notions is pictured in Figure 1.

Simplified theory: definitions

Information concerning f

We associate with a function f a recursively defined evaluation algorithm A and a function table T. f maps an ordered finite domain X of size N into a finite co-domain Y of size M. T is the ordered set of pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ such that the set of the left-hand elements is equal to X and for each pair $f(x) = y$.

If the right-hand elements of T taken in the order $y_1, y_2, \ldots, y_N$ are interpreted as the symbols of a message composed from an alphabet, Y, a natural expression for the amount of information associated with f is

$$I(f) = - N \sum_{Y} p(y) \log \; p(y) \quad \text{bits}, \qquad \ldots\ldots\ldots\ldots\ldots (1)$$

where p(y) is the relative frequency of occurrence of y in the message. We prefer a re-formulation as a sum over the N symbols of the message of the "surprisals" associated with the individual symbols, defined for the i-th symbol as $-\log_2 p(y_i)$. This gives us

$$I(f) = - \sum_{i=1}^{N} \log_2 \frac{n(y_i)}{N} \quad \text{where } n(y_i) \text{ is the number of occurrences}$$

of y_i as a right-hand element of T. This may be rewritten

$$I(f) = N \log_2 N - \sum_{i=1}^{N} \log_2 n(y_i) \text{ bits} \quad \ldots\ldots\ldots\ldots (1A)$$

Knowledge-content of the look-up program p_T

A program p_T implements T as a look-up table for evaluating f on a machine of specification S. Let β be an upper bound to the evaluation time of f for any x in X. Suppose that look-up consumes arbitrarily little time on each occasion and in any case always less than β. With the rôle of β thus rendered vacuous, the system's knowledge of f is equal to its total stored information about f, subject to the aforesaid restriction relative to β. Thus p_T's knowledge-content is

$$K_T = N \log_2 N - \sum_{i=1}^{N} \log_2 n(y_i) \quad \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Note that the numerical value is equal to that of I(f) and also to the store-occupancy in binary digits of a minimal-length version of p_T.

Knowledge-content of the solution program p_A

Now consider a different program p_A for evaluating f. p_A implements A on a machine of the same specification S. Let β-evaluable be a predicate which is true for just those x's for which f(x) can be evaluated within this bound, and let π denote the null program.

Then $X_A = \{x | x \in X \wedge \beta\text{-evaluable}(f, x, p_A, \pi, \pi, S)\}$

and $T_A = \{(x_i, y_i), (x_j, y_j), \ldots, (x_k, y_k)\}$ such that the set of the left-hand elements of T_A is equal to X_A (which has N_A members) and, for each pair f(x) = y. Clearly $X_A \subseteq X$ and $T_A \subseteq T$. We call the function f_A associated with T_A the "algorithmically known" part of f.

p_A's knowledge-content is thus

$$K_A = N_A \log_2 N - \sum_{i=1}^{N_A} \log_2 n(y_i) \quad \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

where N_A is the size of T_A.

Difference between information-content and knowledge-content

Clearly the amount of information associated with A is the same as that associated with T, namely I(f). Yet their respective implementations differ widely in knowledge-content. For K_A to equal I(f) either β must be arbitrarily increased or S so altered that the machine runs arbitrarily fast, allowing the system to solve all problems in X within user-acceptable time.

Note some connexions with the Chaitin-Solomonoff-Kolmogorov definitions (see Chaitin, 1975) of randomness and complexity. If p_A^* is a minimal-length implementation of A, then its store-occupancy in bits, which we shall write $L_{min}(A)$, is equal to what in Solomonoff and Kolmogorov's (Chaitin 1975) terminology would be called the complexity of T, and in ours its ω-complexity (ω stands for an arbitrarily long time-allowance for reconstructing the whole of T from p_A). The maximum attainable compression ratio for T is, in our notation

$$\frac{I(f)}{\omega\text{-complexity}(f)} = \frac{L_{min}(T)}{L_{min}(A)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

where $L_{min}(T)$ is a minimal encoding of T as a look-up table for an abstract machine of the same specification as that on which p_A^* can be executed. We will later see that this ratio defines the maximum computational advantage attainable for the evaluation of f (e.g. by Martians, who think infinitely fast and never need advice).

Computational advantage is defined as an information ratio: information-bits divided by store-bits. It is the number of bits of β-retrievable information per bit of store. We give the name penetration to the logarithm of this ratio. Computation is a trick for exploiting the time dimension to achieve penetration into f, subject to the limitation β. If we optimize a pure look-up system, we get computational advantage equal to unity, zero penetration. Penetration is expressed in "decs" (decimal compressions). "40 3-minute decs", for example, characterizes a system which computes, in less than three minutes for each input, an f whose T would require n bits for explicit storage, but which manages with only $10^{-40}n$. The informational economics of chess are such that a Grandmaster might constitute a realistic example.

Knowledge-content of advice program and knowledge-overhead of control program

On a machine of specification S which runs the solution program p_A, an advice program p_B implements a body of advice B. Making the advice available to p_A requires the action of control routines p_C. In the simplified theory we assume a time penalty associated with the invocation of p_C which imposes an overhead on every attempted evaluation of f.

Consider the ordered sets

$$X_{AC} = \{x | x \in X \wedge \beta\text{-evaluable}(f, x, p_A, \pi, p_C, S)\},$$

and $T_{AC} = \{(x_i, y_i), (x_j, y_j), \ldots, (x_k, y_k)\}$ such that the set of left-hand elements of T_{AC} is equal to X_{AC} and for each pair $f(x) = y$.

Since the invocation of p_C on each call of p_A consumes time, but obtains no return (because the advice program is null), the predicate β-evaluable may give the result *false* for some members of X_A (for all of which it formerly gave *true*). Hence $X_{AC} \subseteq X_A$ and $T_{AC} \subseteq T_A$. Correspondingly

the quantity

$$K_{AC} = N_{AC} \log_2 N - \sum_{i=1}^{N_{AC}} \log_2 n(y_i)$$ is interpretable as the

amount of algorithmic knowledge of f less the knowledge-overhead incurred by p_C.

Now replace the null program by p_B and consider the ordered sets

$X' = \{x | x \in X \wedge \beta\text{-evaluable}(f, x, p_A, p_B, p_C, S)$, which has size N' and $T' = \{(x_i, y_i), (x_j, y_j), \ldots, (x_k, y_k)\}$ such that the set of left-hand elements of T' is equal to X' and for each pair $f(x) = y$.

We form T_B (which has N_B members) as $T' - T_{AC}$ and define the quantity of advisory knowledge of f relative to p_A as

$$K_B = N_B \log_2 N - \sum_{i=1}^{N_B} \log_2 n(y_i) \quad \ldots\ldots\ldots\ldots\ldots (5)$$

The system's overall knowledge of f is measured as

$$K = N' \log_2 N - \sum_{i=1}^{N'} \log_2 n(y_i) = K_A + K_B + K_C \quad \ldots\ldots\ldots (6)$$

where K_C, a negative quantity, is the knowledge-overhead associated with the control program, and is measured as $K_{AC} - K_A$. Note that corresponding to T' is a partial function f_K which we call the known part of f, and that f_K can be further partitioned into f_{AC} and f_B corresponding to T_{AC} and T_B respectively.

Now if the store-occupancies of p_A, p_B, p_C are L_A, L_B, L_C, termed respectively algorithmic cost, advisory cost and overhead, then

$$L = L_A + L_B + L_C = \text{overall store-cost} \quad \ldots\ldots\ldots\ldots (7)$$

$$C_A = \frac{K_A}{L_A} = \text{algorithmic advantage} \quad \ldots\ldots\ldots\ldots\ldots (8)$$

$$C_B = \frac{K_B}{L_B} = \text{advisory advantage relative to } p_A \quad \ldots\ldots\ldots (9)$$

$$C = \frac{K}{L} = \frac{K_A + K_B + K_C}{L_A + L_B + L_C} = \text{overall computational advantage} \ldots\ldots (10)$$

Note that all these measures of advantage can be thought of as compression ratios relative to the smallest store-occupancies which could be attained if f_A, f_B and f_K were represented in pure look-up form.

Let L_β be the β-complexity of f relative to S', i.e. the smallest store cost for which complete knowledge of f could be obtained with a machine specification S' differing from S only in having unlimited store. Let L be the total program space available in S and let $L < L_\beta$. If K^*_L is the maximum amount of knowledge about f attainable for store cost L (not for fixed p_A which we are free to rewrite in any way whatsoever), then

$$\frac{K}{K^*_L} = \text{the computational yield of the system} \quad \ldots\ldots\ldots (11)$$

Another interesting quantity is $\frac{I(f)}{L_\beta}$ which gives the maximum attainable computational advantage for a system limited by the β bound - compare with $\frac{I(f)}{L_{min}(A)}$ cited earlier for Martians.

Consider now K, L_K for some program and define L^*_K as the minimum store in which K units of knowledge about f can be represented, i.e. L^*_K is the β-complexity of the partial function f_K. Then since $\frac{K}{L^*_K}$ is the maximum attainable computational advantage for given K, the computational efficiency, defined as

$$\frac{L^*_K}{L_K} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (12)$$

can be regarded as the ratio of an actual to an ideal advantage, viz.

$$\frac{\frac{K}{L_K}}{\frac{K}{L_K^*}} \quad .$$

Dropping the subscript K to make room, if L_A^* is the length of a minimal p_A representing K_A units of knowledge about f, and L_A is the length of an actual p_A with the same K_A, then

$$\frac{L_A^*}{L_A} = \text{the algorithmic efficiency of } p_A \quad . . . . . (13)$$

Define L_B^* as the minimum store which can contain a program p_B with K_B units of advisory knowledge relative to p_A. Then

$$\frac{L_B^*}{L_B} = \text{the advisory efficiency of } p_B \quad . . . . . (14)$$

A similar quantity, effectiveness, relates p's advantage $\frac{K}{L}$ to the maximum advantage attainable by a program with maximum K (i.e. I(f)) rather than by a program with given K. We then define effectiveness as

$$\frac{K \times L_\beta}{L \times I(f)} \quad . . . . . (15)$$

Applications of this expression are illustrated graphically in Figure 2.

Refinements: partial (truncated) evaluation

If evaluation of f for the i-th member of X is interrupted by the β cut-off before completion then in the foregoing treatment the system was allotted a zero score for its knowledge of $f(x_i)$. This simplification is replaced by the following, in which we no longer speak of the "known part" of f, since some knowledge adheres to every part (note that the treatment

is closely related to Good's (1976) discussion of the "knowledge that $12^3 = 1728$").

Let the evaluation have the form of a successive approximation in which an initial candidate set of results is reduced in succeeding cycles, ending in a unit set consisting of $f(x_i)$ when evaluation goes to completion but otherwise in a residual set of candidates containing $f(x_i)$. The initial candidate set is the entire co-domain Y. In each time-interval a program p transforms this set according to the sequence $Y \to Y_{i,1,p} \to Y_{i,2,p} \to \ldots \to Y_{i,\beta,p}$ such that $Y \supseteq Y_{i,1,p}$ and $Y_{i,j,p} \supseteq Y_{i,(j+1),p}$. $Y_{i,\beta,p}$ is the candidate set at the moment of β cut-off. We require to quantify the information (surprisal) associated with $Y_{i,\beta,p}$ so that we can measure the system's overall knowledge of f as $\sum_{i=1}^{N}$ surprisal $(Y_{i,\beta,p})$.

Where evaluation is always complete, so that $Y_{i,\beta,p} = \{y_i\}$, we have seen that the information gained from evaluating $f(x_i)$ is $-\log_2 p(y_i) = -\log_2 \frac{n(y_i)}{N}$, where $n(y_i)$ is the number of occurrences of the symbol y_i as a right-hand element in T. Where $y_i \in Y_{i,\beta,p}$ and $Y_{i,\beta,p}$ may have more than one member (i.e. the evaluation, although incomplete, is correct as far as it has gone) we write $-\log_2 \sum_{v \in Y_{i,\beta,p}} \frac{n(v)}{N}$. The system's overall knowledge of f is the sum of all the surprisals, namely

$$-\sum_{i=1}^{N} \log_2 \sum_{y \in Y_{i,\beta,p}} \frac{n(y)}{N}$$

$$= N \log_2 N - \sum_{i=1}^{N} \log_2 \sum_{v \in Y_{i,\beta,p}} n(y) \quad \ldots\ldots\ldots\ldots\ldots (16)$$

To individualize the above to the knowledge content of given programs p_T, p_A, p_B, p_C we substitute for p, thus:

$$K_T = N \log_2 N - \sum_{i=1}^{N} \log_2 \sum_{y \in Y_{i,\beta,p_T}} n(y) ,$$

and similarly to construct definitions of K_A, K_B and K, using $p = p_A$, $p = (p_A, p_C)$ and $p = (p_A, p_B, p_C)$.

Refinements: error

In the simplified theory either an evaluation is complete, or it contributes zero knowledge. But now suppose that for some x_i, a program p completes the evaluation of f, but incorrectly, so that $apply(x_i, p) \neq f(x_i)$. Note that in our scheme error can only arise from the action of p_B. p_A is assumed ex hypothesi to be a correct implementation of A.

To calculate the knowledge associated with the application of p to x_i we multiply the surprisal of $f(x_i)$ by a coefficient $-1 \leq g \leq 1$. g is given by a function defined over the space of pairs (f(x),apply(x,p)). The user can supply any function he pleases provided that

(i) for all $x \in X$, $g = 1$ iff $apply(x, p) = f(x)$;

(ii) for all $x \in X$, a "guessing program" p_G (which selects its output at random from among the members of Y with relative frequencies $p(y_1), p(y_2), \ldots, p(y_M)$, $\sum_{i=1}^{M} p(y_i) = 1$) has zero mathematical expectation for g × surprisal $(apply(x, p_G))$.

Regarding the evaluation of a finite function as analogous to the play of a finite one-person game, g is the outcome value ($g = 1$ corresponds to a "win") and $g \times -\log_2 \frac{n(y_i)}{N}$ is the payoff. This terminology differs somewhat from Good's (1976: his Examples 1 - 5) who associates

different "utilities" with correct and erroneous results of function evaluation. We reserve this term (see later) for discrimination between different members of X ("argument-utilities"), and not for discrimination between different possible results of evaluating f(x) for given x.

Simplest case of error: no gradations, no partial evaluation

In the simplest case, the knowledge content for a correct result is $g \times -\log_2 \frac{n(y_i)}{N}$ where $y_i = f(x_i)$ and $g = 1$. An incorrect result is assigned a negative payoff $-g \times -\log_2 \frac{n(y_i)}{N}$, and the value $g = \frac{n(y_i)}{N - n(y_i)}$ satisfies condition (ii), namely that p_G's expected payoff is zero, as can be verified from the following tabulation.

type of result	frequency when computed as apply(x, p_C)	payoff	contribution to expectation (= frequency × payoff)
correct	$\frac{n(y_i)}{N}$	$1 \times -\log_2 \frac{n(y_i)}{N}$	$\frac{n(y_i)}{N} \times -\log_2 \frac{n(y_i)}{N}$
incorrect	$\frac{N - n(y_i)}{N}$	$-\frac{n(y_i)}{N - n(y_i)} \times -\log_2 \frac{n(y_i)}{N}$	$-\frac{n(y_i)}{N} \times -\log_2 \frac{n(y_i)}{N}$

Gradations but no partial evaluation

In the more general case differentiation is made between different degrees of discrepancy between computed and true values. In chess, for example, an attempt to compute bestmove may be thought more of a failure if the computed result changes a won to a lost position than if it changes a win to a draw.

The user again supplies his own penalty function, provided that it conforms to the same constraints, (i) and (ii).

Partial evaluation, no gradations

Now consider error in the case of a truncated evaluation which yields a set $Y_{i,\beta,p}$ of size m_i. This set may or may not contain $f(x_i)$.

To specify the knowledge-content of p, allowing for error, we introduce a guessing program p_G which for each given x_i gives a candidate set also of size m_i. We suppose that p_G proceeds by making random deletions from an initially complete function table T, until just so many (x, y) pairs remain that the union of their right-hand members forms a set of size m_i - the candidate set Y_{i,β,p_G}. Initially there are $n(y_i)$ pairs in T with y_i as right-hand member. The probability that after d successive deletions no pairs of this kind remain can be calculated from the contingency table:

	undeleted	deleted	
right-hand member = y_i	0	$n(y_i)$	$n(y_i)$
right-hand member $\neq y_i$	N - d	d - $n(y_i)$	N - $n(y_i)$
	N - d	d	N

Fisher's "exact" method (Statistical Methods for Research Workers) gives the probability as

$$\frac{d!\ (N - n(y_i))!}{N!\ (d - n(y_i))!}$$

Denoting this quantity by 1 - P, we now construct a penalty function as shown in Table 1.

General case: gradations of error in partial evaluation

In Table 1 g can be interpreted as a proximity with value 1 for zero difference between computed and true values and $-\frac{P}{1 - P}$ for any other difference. Generalizing this we first consider a proximity

type of result	frequency when computed as apply (x, p_G)	payoff	frequency × payoff
Y_{i,β,p_G} contains y_i	P	surprisal (Y_{i,β,p_G})	P × surprisal (Y_{i,β,p_G})
Y_{i,β,p_G} does not contain y_i	1 - P	$-\frac{P}{1-P}$ × surprisal (Y_{i,β,p_G})	- P × surprisal (Y_{i,β,p_G})

Table 1. Penalty function for erroneous evaluation in the case of partial evaluation, no gradations of error. The payoff values shown ensure that the mathematically expected payoff of a "guessing program" (see text) is uniformly zero over all entries of the complete function table T. "surprisal" is calculated as in the earlier section "Refinements: partial (truncated) evaluation."

function $\emptyset$ which in the expression $\emptyset(y, f(x_i))$, where $y \in Y_{i,\beta,p}$, yields a value for g, $-\infty < g \leq 1$. g expresses the "proximity" of y to the correct result. A truncated evaluation which yields a candidate set can in this context be viewed as a question-answerer which produces in response to the question x_i a number of possible answers of varying degrees of correctness. To calculate the corresponding knowledge-content, two alternatives present themselves.

(1) We can summarize the candidate set in terms of the expected value, given this set, of $\emptyset(\text{apply}(x_i, p), f(x_i))$, where $\text{apply}(x_i, p)$ is the output value which would be obtained by running the program p to completion on the input x_i. A statistical approach of this kind demands that the user have some model of the computational process which successively shrinks the candidate set.

(2) We can adopt a "worst-case" position and use a penalty function pf defined by

$$pf(Y_{i,\beta,p}) = \min_{y \in Y_{i,\beta,p}} (\emptyset(y, f(x_i))$$

In other words we take the proximity of the candidate set to the correct result as being given by that of its most distant member.

Refinements: differential argument-utilities

As just shown, different answers to a given question rate different values according to their proximity to the correct answer. The user may also, and independently, attach different values to the receipt of answers to different questions. Argument-utilities will be regarded as fixed - corresponding to the notion of "importance." This simplifying assumption is made for exposition only, since they could in practice perfectly well fluctuate (dependent on the progress of some problem-solving process) and thus act as "relevance" weightings. An example of the latter case would be the evaluation of bestmove for a chess position containing Queens at a stage when the Queens had already been swapped off. For such an x as this the system could afford not to know f(x) for the time being.

Argument-utilities are introduced as weighting coefficients thus:

$$K = \sum_{i=1}^{N} u_i \text{ surprisal } (Y_i).$$

A normalizing constraint demands that the distribution of argument-utilities has unit mean: this relates the amount of "useful" knowledge to the amount of "pure" knowledge. We use the qualified term "pure" knowledge in a case where the u_i's are omitted, or, equivalently, set uniformly equal to unity.

Without prejudice to the user's right to define his own utility function, the natural approach is to assume utilities to be directly proportional to the frequencies with which p is called for the different x_i's (compare Good, 1976).

Some β-independent quantities

If one does not like the idea of imposing all-or-nothing cut-offs after the expiration of β seconds, there are other measures which can be applied to the evaluation of f relevant to the notion of knowledge.. We can introduce a time-base e, the time required for the system to evaluate f for every x in X (the "extraction time"). We then define computational capacity as $\frac{I(f)}{e}$, measured in bits per second. Decomposition into algorithmic and advisory components can be performed as before. Likewise $\frac{I(f)}{Le}$ measures computational (algorithmic plus advisory) power; the units are store-equivalents/sec. Power is thus a dynamic version of advantage.

Some β-independent measures are tabulated below.

Extraction time	Seconds	e
Capacity	Baud (bits/sec)	$\frac{I(f)}{e}$
Power	Store-equivs/sec	$\frac{I(f)}{Le}$
Turnover time	Seconds	$\frac{Le}{I(f)}$

Each of the above is separately measurable for the algorithmic, p_A, and advisory, p_B, components of p - giving "algorithmic capacity" and "power"; and "advisory capacity" and "power".

The harshness of the β cut-off idea can be softened in another way.

Introduce a multiplier r_t which is a function of elapsed time t, and rewrite the expression of the preceding section as

$$K = \sum_{i=1}^{N} r_t \, u_i \, \text{surprisal}\,(Y_i) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (16A)$$

If r is a step function such that $r_t = 1$ for $t \leq \beta$ and $r_t = 0$ otherwise, then K is maximized when $t = \beta$, which corresponds to the earlier treatment. Now let r be some other function, monotonically decreasing in

t. The utility of the result is then a continuously decreasing quantity during the evaluation process, just as the information obtained is an increasing quantity. In this case also we can choose t in such a way as to maximize (16A). An optimal waiting time is thus again derived from a user-supplied utility function, but in a more subtle fashion than in the original simple β cut-off case.

r can of course be made a function not only of time but of the current state of the computation itself - as when it is decided to devote further time to a given evaluation because it has been "proving difficult." But the main exposition will not be served by pursuing such elaborations further.

The memo-function as a model of rote-learning

The most elementary form which the scheme of Figure 1 can take, and one which has received experimental study in the author's laboratory, is the "memo-function" (Michie, 1967). Implementations by Michie, Popplestone and Marsh (1968) and by van Emden (1974) allow the user to endow a function procedure with a private memory in which it automatically stores previously computed results. On each call of the procedure, the rote-memory is first accessed in an attempt to evaluate f by look-up. Only if the rote-search fails is the body of the procedure entered. The aim is to improve evaluation time.

It will immediately be observed that if the procedure's initial knowledge of f is restricted by a β-bound on evaluation time then the introduction of a rote-memory confers an increase of knowledge - precisely measurable in our system. In effect, the memo-function instantiates Figure 1 by substituting for p_R a dynamically varied fragment of p_T. The rote

consists solely of advice at "ground level" - i.e. no abstraction, only selection, from f's function-table. The algorithmic procedure (memo-ised by the library routine "newmemo") corresponds to p_A, while the library package itself consists of p_C (for applying the rote) and p_U (for updating it). With repeated application of p_A to inputs sampled from the function's domain, (x, y) pairs accumulate in the rote and evaluation times decrease. Gains accrue not only when the input argument itself is found to be already on the rote, but also from rote-knowledge of results required in recursive calls: if you want to evaluate factorial (17) it helps if you already know factorial (16) - or even factorial (6). By our definitions, then, β-evaluability increases, i.e. the system's advisory knowledge of f. Note that the advisory cost also increases as the rote grows.

If we "freeze" the rote at some stage, paralyzing the rote-updating mechanism, we can directly apply the earlier techniques to the knowledge associated with p_B at the moment of freezing. But how are we to measure the cost-effectiveness not of the behavior of the program as a device for knowing about f (we have seen how to do this) but its behavior as a device for getting to know more about f? The following approach seems workable.

Measurement of an amount of knowledge is concerned with properties of (p_A, p_B, p_C). Measurement of rates of increase of the amount is concerned with properties of p_U, the update program, in particular its capacity to enhance the value to the user of p_B. For a given p_U we want to have some measure of its effectiveness in doing this, and of the cost incurred.

First we must decide precisely what attribute of p_B to use in the numerator of p_U's ratio of effectiveness to cost. Clearly it should be related to the increase of p_B's advisory advantage conferred by p_U: a p_U conferring a small knowledge-gain but a large cost-gain upon p_B must not

be allowed to rate high on effectiveness. We have chosen the logarithm of advisory advantage, i.e. advisory penetration, expressed in decs ("decimal compressions"). We shall use the symbol D_B for this quantity.

The denominator must be a function of p_U's store-cost and the time spent by the processor in p_U, both of which plainly contribute to the total cost of running p_U over the learning period. We must also consider the costs associated with running p_A in conjunction with p_B and p_C to provide rote entries in the first place. We accordingly define a learning index:

$$R = \frac{\Delta D_B}{L_U t_U} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (17)$$

L_U is the store-cost of p_U, and t_U is the time spent in p_U by the processor during the learning period. The meaning of R is "increase in advisory penetration conferred by the operation of p_U, relative to the cost of its operation." The concomitant operation of the rest of the program (p_A, p_B, p_C) also has a cost, which can be thought of as the cost of the "practice evaluations" separate from the learning cost.

We have devoted space to the economics of rote-learning, because we shall later require to do the same for the economics of "concept-learning" - i.e. to measure the capability of a given p_U to enhance p_B's value to the user in the case where p_B is not the rote of a simple memo-function, but contains more organized information abstracted from f's function-table. We shall then use the rote-learning formulation as a base-line for measurement of the ability to acquire conceptualized as opposed to merely rote knowledge.

Telegram-factory model of concept-learning

A concept learning system is a device for the synthesis of cheap but effective inter-galactic telegrams. The stimulus for this deliberately mysterious remark is a passage from G. J. Chaitin (1975):

> *"Suppose you have a friend who is visiting a planet in another galaxy, and that sending him telegrams is very expensive. He forgot to take along his tables of trigonometric functions, and he has asked you to supply them. You could simply translate the numbers into an appropriate code (such as the binary numbers) and transmit them directly, but even the most modest tables of the six functions have a few thousand digits, so that the cost would be high. A much cheaper way to convey the same information would be to transmit instructions for calculating the tables from the underlying trigonometric formulas, such as Euler's equation* $e^{ix} = \cos x + i \sin x$. *Such a message could be relatively brief, yet inherent in it is all the information contained in even the largest tables."*

The mathematical work which Chaitin is here popularizing, carried out by himself, by R. J. Solomonoff and by A. N. Kolmogorov, has been concerned with measuring the "complexity" of a message as the minimal number of store-bits required to hold a program capable of reconstructing it. In this theory, the inter-galactic telegram is to be run as a program on the receiver's machine. Provided that it terminates in finite time no one cares how long it takes to perform the computation.

But consider the following scenario, diagrammed in Figure 3. Our distant friend has to play in a chess tournament, and asks us to send him, not the trigonometric functions, but the bestmove function for chess. We may suppose that the tournament rules demand, as on Earth, that moves be made without undue delay. Neither of the following two responses to his request will earn any marks:

(1) to send the complete look-up table, p_T in our notation. The total bit-cost of the telegram will be in the region 10^{50}.

(2) to follow the Chaitin prescription and send a minimal program embodying an evaluation algorithm (in our case say the classic Zermelo-Borel-von Neumann total-lookahead algorithm), a minimal p_A in our notation.

In the case of (2) all the information has certainly been sent - at a cost of only about 10^4 bits. But our friend will object that we have not put him in possession of much useful knowledge. What should we have sent him? Plainly a telegram comprising p_A and p_B and p_C. An argument based on the maximum rate of transfer of information into human long-term memory suggests that a Grandmaster is unlikely to have acquired more than 10^{10} bits of chess information for his p_B. So although the telegram will be costly compared to one consisting of p_A alone, its cost is a drop in the ocean of p_T's cost. But like p_T, and unlike unaided p_A, it is executable within acceptable waiting time.

We now ask: "What is the shortest inter-galactic telegram which will enable the recipient to β-evaluate bestmove for every x in X?" By obvious analogy with Chaitin-Solomonoff-Kolmogorov complexity (which we refer to as ω-complexity) we term the length of the shortest such telegram the β-complexity of bestmove (always of course with respect to S). In contrast with ω-complexity, the interest of which is purely mathematical, we find in β-complexity an explication of the notion of intrinsic intellectual difficulty as a measurable (in principle) property of a problem domain. The "bestmove" function for a large version of the game of Nim, say with 16 heaps of 1000 matches each, has approximately the same information-content as for chess. But the corresponding p_A (implementing the well-known parity rule of play) is so fast to evaluate that β-complexity can hardly differ from ω-complexity and is small. So 16 × 1000 Nim has little

intrinsic difficulty. But here is the continuation of the previously quoted passage from Chaitin.

> *"Suppose, on the other hand, your friend is interested not in trigonometry but in baseball. He would like to know the scores of all the major-league games played since he left the earth some thousands of years before. In this case it is most unlikely that a formula could be found for compressing the information into a short message; in such a series of numbers each digit is essentially an independent item of information, and it cannot be predicted from its neighbors or from some underlying rule. There is no alternative to transmitting the entire list of scores."*

In this case ω-complexity is high, and a fortiori so is β-complexity. Answering questions about baseball scores would be, at the least, a difficult task.

Escalating the inter-galactic fantasy, suppose now that appeals from our distant friend for this or that finite function become so frequent that consideration must be given to the economics of automatically synthesizing new telegrams to his requirements. The resulting set-up is depicted in Figure 4. The recipient's machine, as before, is cost-free but the sender's machine is not, and must operate as a cost-effective telegram-factory. For each new synthesis the sender machine starts with p_A, p_C and some p_B (possibly null) ready loaded. It is thus already in a position to emit either of two unsatisfactory answers: to transmit a telegram consisting merely of p_A, or to run p_A over the whole domain in order to generate and send p_T. The first would be good for the sender, bad for the receiver; the second good for the receiver, bad for the sender. To produce the right answer the brunt of the sender machine's task falls on p_A and p_U, respectively to explore fragments of f ("practice") and to generalize over p_A's discoveries to build a body of advice p_B. At some stage a halt is called, and the telegram (p_A, p_B, p_C) is transmitted. p_U stays at home. The

total cost is the manufacturing cost plus the cost of inter-galactic transmission. Manufacturing cost in turn decomposes into learning cost (Lt for runs of p_U) and cost of practice (Lt for the associated runs of (p_A, p_B, p_C)). An illustrative miniature example of a p_U operating in something like this style is Negri's (1976) use of an inductive learning program to synthesize a p_B able to discriminate won from drawn positions in the K + P v. K chess end-game. One envisages its use coupled with a p_A which looks ahead until positions of known game-theoretic values are encountered. Lookahead would initially go to the end of the game. The final telegram could be the multivariable logic expression induced by the program.

Learning, rote-skill and grasp

To justify on economic grounds the addition of an advisory package (p_B, p_C) to a solution program it is not enough, or even to the point, to show that the package confers increase of knowledge: it must confer enhanced computational advantage or (to say the same thing logarithmically) enhanced penetration. Note that we earlier defined learning in terms of increase of penetration over successive trial evaluations, not in terms of increase of knowledge. Even a decrease of knowledge, if accompanied by a disproportionate decrease in p_B's bit-cost, could qualify as a form of learning in our scheme. The critical inequality, which justifies a given modification of p_B, or indicates that a step of positive learning has occurred, is

$\frac{K_{B'} \times L_B}{L_{B'} \times K_B} > 1$, or equivalently $D_B < D_{B'}$, where B relates to the advice-program p_B and B' to its modified form. The ratio on the left above

is the "relative advisory effectiveness" of p_B, with respect to p_B.

With this in mind, let us dissect advisory advantage into two dissimilar components, one associated with a rote-dictionary in the style of a memo-function and the other associated with a "concept-dictionary," in which certain of the (x, y) "facts" discovered in past evaluations have been condensed into descriptional form. The "ground-level" is distinguished from the "conceptualized" component:

$$\text{Rote advantage} = \frac{K_r}{L_r} \quad ; \quad \text{conceptual advantage} = \frac{K_c}{L_c}$$

We shall call the logarithms of these quantities "rote-skill" and "grasp" respectively. For some purposes it will be useful to write advisory advantage in the form

$\frac{K_r + K_c}{L_r + L_c}$, especially when considering the profit-and-loss score of assimilating rote elements into the concept-dictionary.

A consequence of the foregoing system of definitions is that an ordinary recursive procedure for the factorial function is credited with penetration but with neither rote-skill nor grasp. The memo-ised version of the same procedure can claim rote-learning, defined in our system as the autonomous improvement of penetration specifically by increase of rote-skill. Improvement through use of the memo-function's facility for direct assignment of values by the user to the rote exemplifies teaching by rote. If after some regime of rote-learning and/or rote-teaching we decide to suspend the action of p_U and to investigate whether the memo-function in the form in which it has been frozen shows signs of having profited from

the regime, then we shall pronounce positively only if we find that penetration has indeed been increased beyond the base-line set by p_A. If additionally the memo-function's up-date program p_U were of itself to impose a beneficial structural reorganization upon the rote, for example by introducing dynamic grouping of result-values into intervals as in Marsh (1969) or (to be fanciful) by discovering Stirling's approximation and using it to replace in p_B all values larger than a certain threshold by the corresponding predicate, then and only then could we say not only that this particular (p_A, p_B, p_C) exhibited penetration in the form of rote-skill and grasp, but that the associated p_U had exhibited concept-learning.

A recapitulation of some of these ideas is set out in Table 2.

Quality or aptitude	Informal explication
Knowledge	Retrievable information.
Advantage	Ability to retrieve more than is stored.
Penetration	Logarithm of advantage.
Rote-skill	Penetration attributable to storage of rote-facts.
Grasp	Penetration attributable to storage of descriptions.
Difficulty (β-complexity)	Size of smallest program able to confer complete knowledge.
Rote-learning	Increase of rote-skill by storing results of past evaluations.
Parameter-learning	Increase of grasp by adjusting already stored descriptions in the light of past evaluations.
Concept-learning	Increase of grasp by abstracting new descriptions from past evaluations.

Table 2. Recapitulation of some main ideas. An intermediate level of learning ("parameter learning," not discussed in the text) has been inserted. Examples of this mode are found in early AI work such as that of Samuel (1959) and of Selfridge (1959), and in many modern adaptive control systems. A "description" is an intensional definition of a (typically non-unit) set.

Concluding remarks

In the pre-scientific era, lack of a system of numerical definitions concerning mass and motion hindered the development of the mechanical arts. An analogous situation prevails today with regard to machine intelligence and provides incentive for the present work. Detailed mathematical and experimental development is required. The latter could usefully include the programming of a cluster of interrelated subdomains (for example chess end-games) using an invariant p_A. Knowledge and learning parameters of

different p 's and p 's could thus be systematically compared over a range of cases.

Acknowledgements

I am indebted to I. J. Good, Martin Kruskal, David Plaisted and Boris Tsybakov for helpful discussion and in particular to Rod Fletcher, John McCarthy and Michael Tanner who spotted errors in earlier versions. The Department of Computer Science at the University of Illinois provided me with facilities for a substantial part of the work. I wish in particular to thank Pam Farr who was responsible for meticulous typing of difficult material.

References

Buchanan, B. G. (1975) Applications of Artificial Intelligence to scientific reasoning. Proc. 2nd USA-Japan Computer Conference.

Chaitin, G. J. (1975) Randomness and mathematical proof. Scientific American, 232, 47-52 (May, 1975).

Good, I. J. (1976) Dynamic probability, computer chess, and the measurement of knowledge, in Machine Representations of Knowledge (eds. E. W. Elcock and D. Michie), Dordrecht: D. Reidel Publishing Company.

Hartley, R. V. L. (1928) Transmission of information. Bell System Tech. Jour., 7, 535-563.

Huberman, B. J. (1968) A program to play chess end games. Technical report no. CS 106. Stanford University: Computer Science Department.

Marsh, D. L. (1969) Memo functions, the Graph Traverser and a simple control situation, in Machine Intelligence 5, pp. 281-300 (eds. B. Meltzer and D. Michie), Edinburgh: Edinburgh University Press.

Meltzer, B. (1970) The semantics of induction and the possibility of complete systems of inductive inference. Artificial Intelligence, 1, 189-192.

Michalski, R. and P. Negri (1976) An experiment on inductive learning in chess end games: the King-Pawn-King case, in Machine Representations of Knowledge (eds. E. W. Elcock and D. Michie), Dordrecht: D. Reidel Publishing Company.

Michie, D. (1967) Memo functions: a language facility with "rote-learning" properties. Research Memorandum MIP-R-29, Edinburgh: Department of Machine Intelligence and Perception.

Michie, D., Popplestone, R. J., and Marsh, D. L. (1968) LIB MEMOFNS. pp. 209-212 of Programming in POP-2 by R. M. Burstall, J. S. Collins and R. J. Popplestone. Edinburgh: Edinburgh University Press, 1971.

Michie, D. and B. G. Buchanan. Artificial Intelligence in Mass Spectroscopy: a review of the Heuristic Dendral program. In Computers for Spectroscopists (ed. R. A. G. Carrington) London: Adam Hilger.

Samson, E. W. (1951) Fundamental natural concepts of information theory. Air Force Cambridge Research Station: Report E5079.

Samuel, A. L. (1959) Some studies in machine learning using the game of checkers. IBM Jour. Res. Dev., 3, 211-229.

Selfridge, O. G. (1959) Pandemonium: a paradigm for learning. NPL Symposium no. 10, Mechanization of Thought Processes, 511-531. London HMSO.

van Emden, M. H. (1974) LIB MEMO FUNCTIONS: program specification. POP-2 Program Library. Edinburgh: School of Artificial Intelligence.

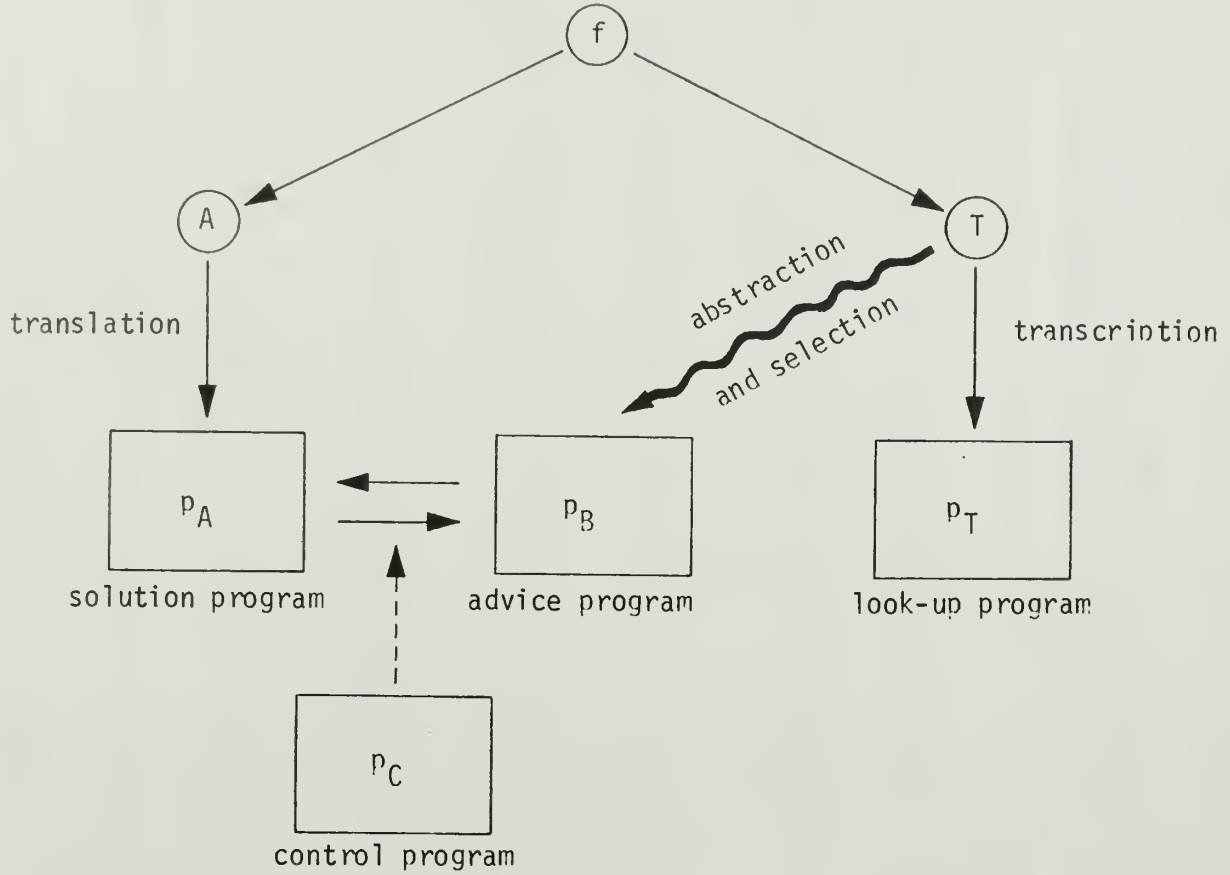


Figure 1

Relations among various kinds of mathematical and computational objects. Abstract objects are ringed, concrete are boxed. A and T denote two contrasted abstract representations of the function f, namely as an evaluation algorithm and as a function table (ordered set of pairs) respectively.

Figure 2

Graphical plots of the relation between knowledge and bit-cost for programs of contrasted characteristics:

	bit-cost	knowledge
p^* - idealized program (minimal length, complete knowledge)	L_β,	$I(f)$
p_1 - non-minimal program with complete knowledge	L_1,	$I(f)$
p_2 - program with incomplete knowledge	L_2,	K_2
p_3 - another program with incomplete knowledge	L_3,	K_3
p_4 - minimal look-up program with complete knowledge	$I(f)$,	$I(f)$
p_5 - non-minimal look-up program with complete knowledge	L_5,	$I(f)$

For a given p, the tangent of the angle made by the broken line with the horizontal axis measures the program's computational advantage.

Effectiveness of the i-th program $p_i = \dfrac{K_i\, L_\beta}{L_i \times I(f)}$

i.e. the computational advantage of p_i relative to that of p^*.

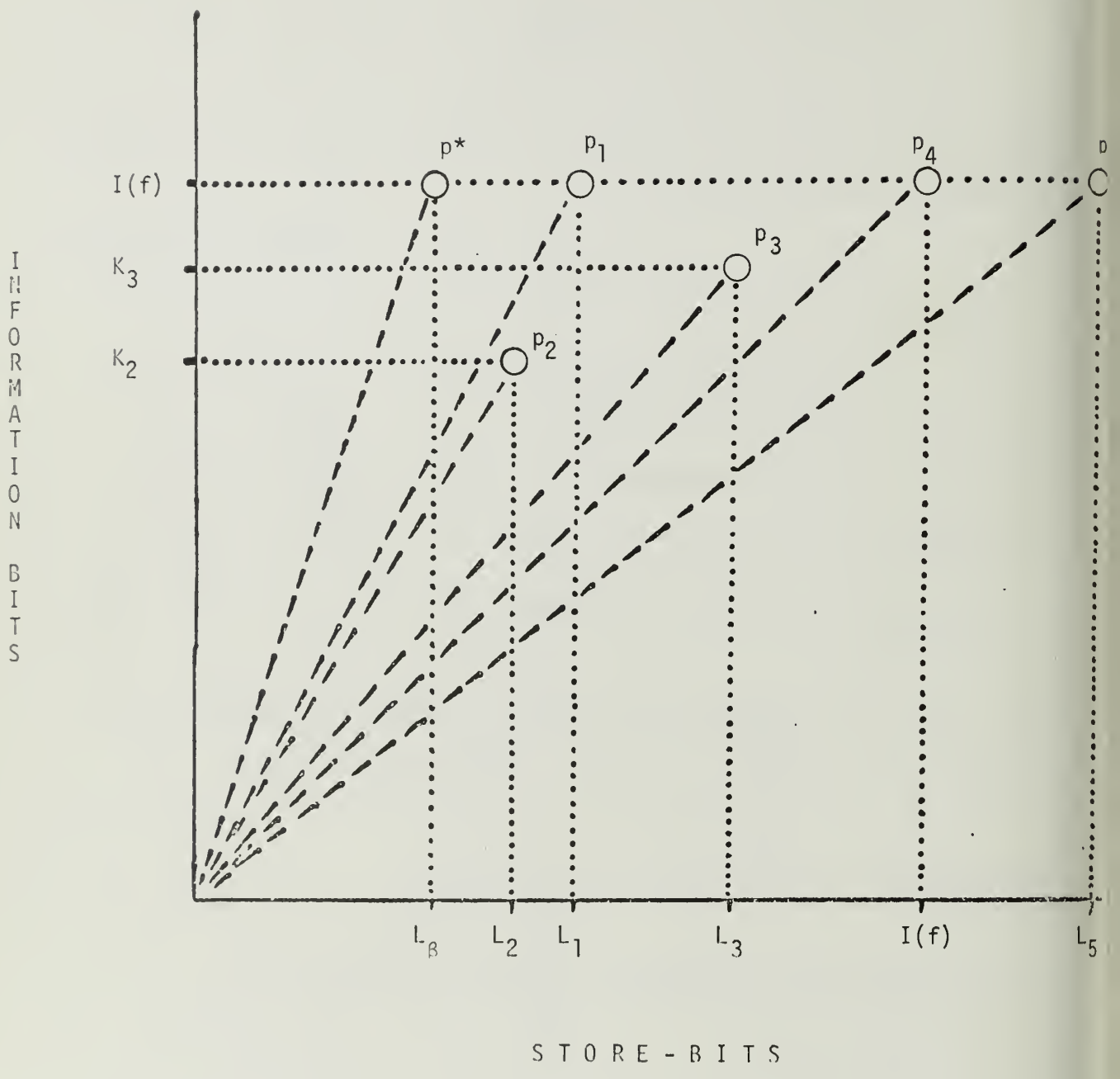

p*
p1
p4
I(f)
K3
p3
K2
p2
INFORMATION BITS
Lβ
L2
L1
L3
I(f)
L5
STORE-BITS

Figure 3

Chaitin's inter-galactic telegram in the context of "advice theory". We allow the sender a cost-free machine of his own in case he wishes to compile his telegram from some specification language (see Chaitin's discussion) and we assign the receiver's machine unlimited data and work-space, and zero processing costs. But we assume a limit of β seconds on the evaluation of f in the TGM for any x. Attention is thus restricted to the bit-cost of the telegram itself.

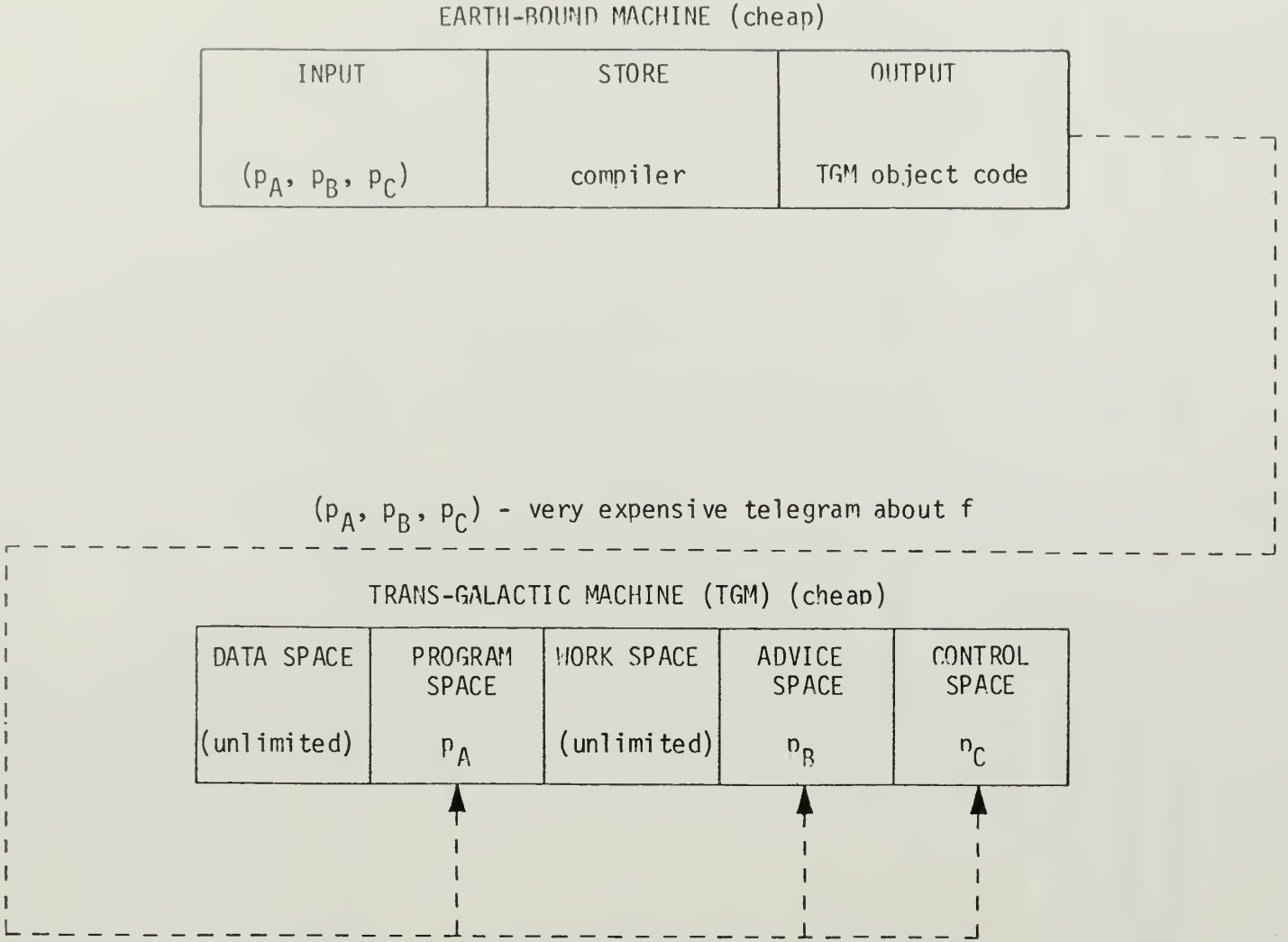
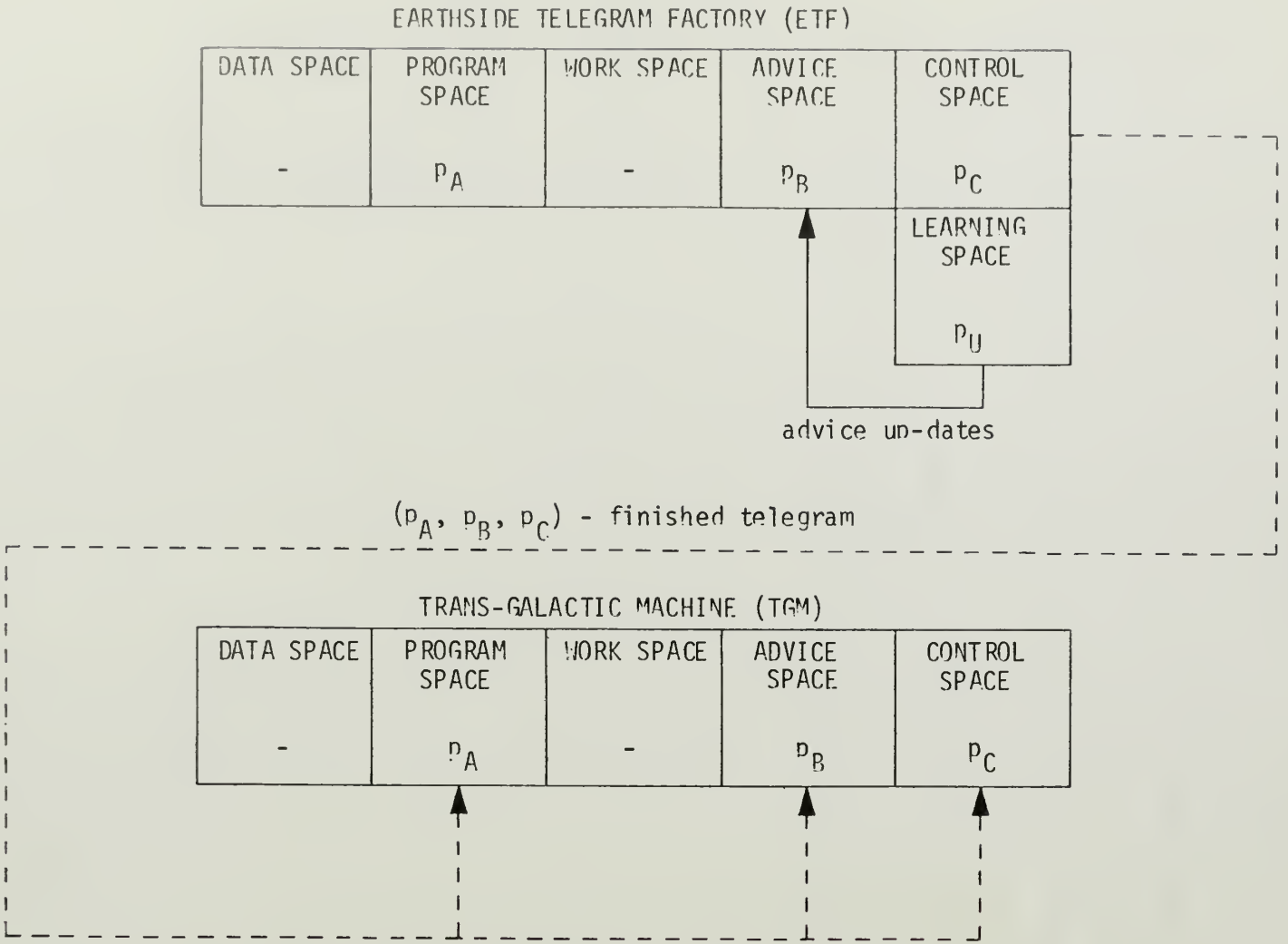

EARTH-BOUND MACHINE (cheap)
INPUT
STORE
OUTPUT
(p_A, p_B, p_C)
compiler
TGM object code
(p_A, p_B, p_C) - very expensive telegram about f
TRANS-GALACTIC MACHINE (TGM) (cheap)
DATA SPACE
PROGRAM SPACE
WORK SPACE
ADVICE SPACE
CONTROL SPACE
(unlimited)
p_A
(unlimited)
p_B
p_C

Figure 4

"Telegram factory" model of conceptual learning. As before, we neglect the costs associated with the TGM. Earthside, there are two categories of cost - the transmission cost of the finished telegram (proportional to its length) and the cost of synthesizing it in the first place. The latter is reckoned under two headings, (i) the storage and running costs of p_{U} - i.e. the "learning" process, and (ii) the storage and running costs associated with the ETF's trial runs of (p_A, p_B, p_C) ("practice").


EARTHSIDE TELEGRAM FACTORY (ETF)
DATA SPACE
-
PROGRAM SPACE
p_A
WORK SPACE
-
ADVICE SPACE
p_B
CONTROL SPACE
p_C
LEARNING SPACE
p_U
advice up-dates
(p_A, p_B, p_C) - finished telegram
TRANS-GALACTIC MACHINE (TGM)
DATA SPACE
-
PROGRAM SPACE
p_A
WORK SPACE
-
ADVICE SPACE
p_B
CONTROL SPACE
p_C

Appendix 1. Relation to Aristotle's theory of knowledge

In the opening passage of the Analytica Posteriora (McKeon, 1947) Aristotle rejects the explicit/implicit dichotomy (his distinction between "unqualified" and "virtual knowledge") as the defining criterion of what is "known" and he incorporates in his definition the case where a man "knows" a fact by prior knowledge of the premisses from which it can be demonstrated. In this case, he says, the man "knows not without qualification but only in the sense that he knows universally"; and he characterizes the man's (implicit) knowledge of the particular fact as "virtual" knowledge. The fact only becomes "known" in the unqualified sense (i.e. explicitly) at the moment that it has first been recognized as a candidate for demonstration from the premisses and has then been successfully demonstrated. Aristotle thus envisages a definite moment of transition from "virtual" knowledge of a fact to "unqualified" knowledge of it, reminiscent of a memo-function's rote-storage of a computed result.

He asks at what stage the student who already knows that the angles of every triangle sum to two right angles can be said to know this also of some particular triangle presented to him: "only," he answers, "at the actual moment at which he was being led on to recognize this as true in the instance before him.... Before he was led on to recognition, or before he actually drew a conclusion, we should perhaps say that in a manner he knew, in a manner not."

Applying this to the example given at the outset of this paper, I know "universally" what numbers are prime through knowledge of the definition of a prime number, I know "virtually" that 233 is prime, and I know "without qualification" that 23 is prime.

The present classification has ventured beyond Aristotle's in two respects:

(1) discussion is not restricted to knowledge of truth-functions, but to finite mathematical functions more generally;

(2) the cut separating "known" from "not known" is placed in such a way as to include in the "known" only those facts which are feasibly demonstrable, i.e. those which the knower can demonstrate from the premisses within an acceptable waiting time. I may reasonably claim "knowledge" that 233 is prime but not (unless I am a calculating prodigy) that 23, 333, 333, 333 is prime. We thus refine Aristotle's system so as to isolate and exclude a sub-category of "virtual knowledge", namely that which is "virtually" but not "feasibly" known (i.e. ω-computable but not β-computable). In our system such material is regarded as definitely not known, although we can only establish in each case whether it is known or not by asking the corresponding question and seeing whether retrieval occurs within our time limit. Aristotle was evidently uneasy about the status of a "virtually" known fact when he said that before its possessor had explicitly derived it "we should perhaps say that in a manner he knew, in a manner not." Probably he did not entertain the possibility that a derivation might be theoretically but not practically executable. By introduction of the β criterion the grounds for unease are resolved, and the ambiguous-seeming "virtual" category is partitioned into "known" and "unknown" sub-categories. In a dynamically evolving knowledge system transmigration of elements may occur in either direction between sub-categories as a consequence of changes in the remainder of the knowledge-base. These can affect in one way or another the total derivation process required for retrieval: new knowledge

can shorten the given fact's retrieval time so that what was previously not known becomes known; or it can lengthen it by adding to its derivation overhead, with the consequence that a formerly known fact can become unknown.

Reference

McKeon, R. (1947) Introduction to Aristotle. New York: Random House (in "The Modern Library" series).

Appendix 2. What to substitute for S.

The machine-dependence of the way in which knowledge is measured may be perceived as an obstacle to mathematical development of the theory. To those wishing to instantiate the machine specification S with something more abstract than, say, "the TENEX operating system mounted on a DEC-System 10 computer with a KI processor, 512K 32-bit words of 1 microsecond store, ... etc." and at the same time to become free to consider the evaluation of finite functions of arbitrary size, then a formalization such as the random access machine (RAM) of Aho, Hopcroft and Ullman (1974) offers attractions. β would then be interpreted as a number of computational steps rather than of time-units.

Reference

Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974) The Design and Analysis of Computer Algorithms. Addison-Wesley Publishing Company, Chapter 1·2.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-76-786	2.	3. Recipient's Accession No.
4. Title and Subtitle MEASURING THE KNOWLEDGE-CONTENT OF PROGRAMS			5. Report Date May 1976 6.
7. Author(s) Donald Michie			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, IL 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No.
12. Sponsoring Organization Name and Address			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

By an application of information theory we measure the information-content of finite functions. The information associated with that subset of a function f which is practically evaluable by a given program on a given machine is identified with the program's "knowledge" of f. By "practically evaluable" we mean that the "answer" y to the "question" f(x) can be retrieved within some resource-bound β. For a real-world machine β is measured in time-units and for an abstract machine in computational steps. Knowledge is thus the information-content of the partial function f_K corresponding to the β-evaluable subset of f. Applications to the notions of "learning" and "difficulty" are discussed.

17. Key Words and Document Analysis. 17a. Descriptors

information theory
information-content
knowledge

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Unlimited	UNCLASSIFIED	57
	20. Security Class (This Page) UNCLASSIFIED	22. Price

OCT 17 REC'D

BOUND BY HNM

UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no.762-787(1976
Design of Irredundant MOS Networks : a p
3 0112 088402588